Web Applications

Software Engineering 2017
Alessio Gambi - Saarland University

Based on the work of Cesare Pautasso, Christoph Dorn, Andrea Arcuri, and others
ReCap
Software Architecture

A software system’s architecture is the set of principal design decisions made about the system.

N. Taylor et al.

<table>
<thead>
<tr>
<th>Abstraction</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualization and Representation</td>
<td>Quality Attributes</td>
</tr>
</tbody>
</table>
Every system a software architecture has

Prescriptive Architecture

Intent

System Artifacts

Realization

Descriptive Architecture

Recovery

What designers want
Design

- Architectural Styles
- Architectural Patterns
- Building Blocks
  - Software Components
  - Component API/Interfaces
  - Software Connectors
Send HTTP request, and get HTTP response containing the HTML page

Browser visualizes it
Client/Server

- Many clients, active, close to users
- One server, passive, close to data
- Single point of failure, scalability
- Security, scalability
HTTP

The Hypertext Transfer Protocol
HTTP

The Hypertext Transfer Protocol

Connector or Component ?
HTTP

The Hypertext Transfer Protocol

Connector or Component?

Synch or Asynch?
HTTP

The Hypertext Transfer Protocol

Connector or Component?

Synch or Asynch?

Stateful or stateless?
HTTP Request

• Action: verb, express the intent

• Headers: meta-data

• Body: optional, can be anything, a stream of bytes, form data, session information, etc.
HTTP Actions

Have precise semantic, and a web application might not implement all of them.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>retrieve a resource</td>
</tr>
<tr>
<td>POST</td>
<td>send data and/or create a resource</td>
</tr>
<tr>
<td>PUT</td>
<td>replace an existing resource</td>
</tr>
<tr>
<td>DELETE</td>
<td>delete a resource</td>
</tr>
<tr>
<td>HEAD</td>
<td>retrieve HEADers but not body</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>check the methods available on the resource</td>
</tr>
</tbody>
</table>

Web applications must to implement the semantic right.
HTTP Response

Headers and status

Status codes, organized in families:
- 1xx: information
- 2xx: success
- 3xx: redirection
- 4xx: user error
- 5xx: server error

Delivers a resource: a page HTML, a CSS file for the style, images, JS libraries, etc.
HTTP Body

• Transfer the main part of the data, but not the only way to send data
  
  *query params, custom headers*

• Required in POST and PUT requests

• Required in responses to GET requests

• HEAD must not provide one
The **Hypertext Markup Language**

```html
<html>
  <head>
    <title>Software Engineering Chair (Prof. Zeller) – Saarland University</title>
    <meta name="description" content="Software Engineering Chair, Saarland University, Saarbruecken, Germany."
    <meta name="keywords" content="software engineering, software analysis, software testing, software debugging, software design">
    <meta name="resource-type" content="document">
    <meta name="distribution" content="global">
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <link rel="StyleSheet" href="/include/style.css" type="text/css" media="screen">
    <link rel="StyleSheet" href="/include/style.css" type="text/css" media="print">
    <link rel="shortcut icon" href="/favicon.ico">
  </head>
  <body bgcolor="#ffffff"
    text="#000000"
    link="#0000ff"
    vlink="#ff0000"
    alink="#ff00ff">
    <table cellspacing=2 cellpadding=0 border=0 width="100%">
      <tr>
        <td valign="middle" align="left">
          <a href="/"><img alt="" src="/logos/stlogo.png" border=0></a>
        </td>
      </tr>
    </td>
    <td valign="middle" align="left" width="50%"><h1><strong class="large_heading">Software Engineering Chair</strong></h1>
      <br>
      <font>...
Resources

One request per resources, multiple requests in parallel
All requests must complete before a page is fully displayed
Static vs Dynamic Pages

Static:

Files are served as they are (index.html)
content does not change
Static vs Dynamic Pages

Static:

Files are served as they are (index.html) content does not change

Dynamic:

The HTML (or part of it) is generated upon request based on data
State-Logic-Display

cluster elements that change at the same rate
Server-Side HTML Rendering

• The HTML page is created on the server and sent back to the client

• Overhead in processing each request if the page is created from scratch

• Same content for different displays
  *Desktop vs Tablet vs Mobile*
Presenter-View

extract the content from the model to be presented from the rendering into screens/web pages
Server-Side HTML Rendering

- Based on **HTML templates** that mix together HTML tags, data, and code

```
<html>
<head>
    <title>Template Page</title>
</head>
<body>
    <h1>Supplies</h1>
    <ul>
        <li>mop</li>
        <li>broom</li>
        <li>duster</li>
    </ul>
</body>
</html>
```

```
let data = {
    title: 'Cleaning Supplies',
    supplies: [
        'mop',
        'broom',
        'duster'
    ];
}

html = new EJS({url:'template.ejs'}).render(data)
```

[http://www.embeddedjs.com/getting_started.html]
Server-Side HTML Rendering

- Based on **HTML templates** that mix together HTML tags, data, and code

- Different technologies:
  
  - *PHP scripts* — *index.php*
  - *JavaServer Faces (JSF)* — *index.xhtml*
  - *Embedded Ruby (ERB)* — *index.html.erb*
Server-Side HTML Rendering

- Based on **HTML templates** that mix together HTML tags, data, and code
- Different technologies:
  - PHP scripts — index.php
  - JavaServer Faces (JSF) — index.xhtml
  - Embedded Ruby (ERB) — index.html.erb
- Templates do not necessarily target the entire page and they might not be stored in “files”
Components

- html.tpl.php
  - page.tpl.php
    - region.tpl.php
      - region.tpl.php
        - node.tpl.php
          - comment-wraper.tpl.php
            - comment.tpl.php
        - region.tpl.php
      - region.tpl.php
    - region.tpl.php
  - region.tpl.php
  - region.tpl.php
    - block.tpl.php
Content Management Systems (CMS)
Joomla! CMS Architecture
Let's take a look under the hood ....
Design
3-Tier Architecture

Presentation

Web Browser

Front End

App Server

Data source

Back End

Business logic
Layered (Style)

- Communications 1 layer up/down
- Information hiding, no circular deps
- Possibly bad performance
- Good evolvability
What run where?

thin-client  

fat-client
3-Tier Architecture

Presentation

Business logic

Data source

Web Browser

Front End

App Server

Back End
Data Layer

- Persistence
- Storage
A Mapping Problem

Domain model

Data storage
Domain Model

- Represent concepts in the domain and their relations, not as rows in a database
- Network of interconnected concepts
- Abstract Data Type
  data and the behavior
Storage model

How to store data?

• Key-Value Model
  list of keys and values (hashtable style)

• Relational Model
  traditional SQL model

• Document-oriented Model
  schema-less documents

• Graph-oriented Model
  data is stored as an interconnected graph
Key-Value

- Implement a map
- Values have no schema
Relational

• Set-theory

• Collection of tables with rows and columns
Document-oriented

- Data stored in document but not relations
- Extends Key-Value
Graph-oriented

- Data stored as network graph
- Relations first-class citizens
Polyglot Persistence

Multiple ways to store data

Financial Data → Relational
Product Catalog → Document
Recommendation → Graph

Relational → Reporting
Document → Key Value
Graph → Shopping Cart
User Sessions
3-Tier Architecture

Presentation

Business logic

Data source

Web Browser

Front End

App Server

Back End
Application Layer

- Data access, navigation, and persistence
- Data processing (Business logic)
Plugin

- Explicit extension points
- Static/Dynamic composition
- Low security (3rd party code)
- Extensibility and customizability
A Different Problem (?)

Conceptual mismatch

Programming Language

Objects

Native Database Structure (e.g., relations)

Storage Architecture

Diagram:
- **Customer**
  - name: String
  - location: String
  - sendOrder()
  - receiveOrder()

- **Order**
  - date: Date
  - number: String
  - confirm()
  - close()

- **SpecialOrder**
  - date: Date
  - number: String
  - confirm()
  - close()
  - dispatch()

- **NormalOrder**
  - date: Date
  - number: String
  - confirm()
  - close()
  - dispatch()

- **Super class**
- **Generalization**

Diagram:
- **Form1**
  - ID
  - Name
  - pID
  - fID

- **Table1**
  - parent_table (persons)
  - relation_table
  - child_table (fruits)

- **Window**
  - Title: Form1
  - Components:
    - TreeView
    - ListView
  - Buttons:
    - New
    - Open
    - Save
    - Close

- **Database**
  - Tables:
    - Persons
    - Orders
    - Products
  - Relationships:
    - Person-Order
    - Order-Product
Data Access API

- Add an abstraction layer that represent the database in the application
- Wrap the communication with the data store and expose it as domain model
Data Source Patterns

• Row Data Gateway
  *One instance per row returned by a query*

• Table Data Gateway
  *One instance per table*

• Active Record
  *Encapsulates DB access and adds business logic to data*

• Data Mapper
  *loads DB into Domain Model, and vice-versa*
Row and Table Gateways

- Based on table structure
- Conversion of object type to database format
- Typically stateless
- Push back and forth data
Active Record

Row Data Gateway + Business Logic

- **Product**
  - productId
  - productName
  - static get(int id)
  - insert
  - update
  - delete
  - calculateRecognitions()

- Methods for:
  - Create instances from SLQ results
  - Insert new instances in the data store
  - Update data store based on instances data
  - Find relevant instances
Data Mapper

- Decouple objects structure from database structure
- There may be more than one mapper per domain object
Navigate (Relational) Data

Core Problem

Conceptual mismatch

Programming Language

Objects

Native Database Structure (e.g., relations)

Traverse object graph

Join over foreign keys
Lazy Loading
Lazy Loading

Interrupt the load at some point and resume it later only if needed
Lazy Loading Patterns

- **Lazy Initialization**
  
  Checks if field is null at every access

- **Value Holder**
  
  Wraps lazy-loaded objects

- **Virtual Proxy**
  
  Mocks field access and loads values on the demand

- **Ghost**
  
  Real object but partially loaded, missing data loaded on first access
3-Tier Architecture

Presentation

- Web Browser
- Front End
- App Server

Business logic

Data source

Back End
Front End

- Generate the HTML based on request and data from the backend
- Can handle client side interactions both inside the server and the client’s browser
- Security, input validation, responsiveness, etc.
<table>
<thead>
<tr>
<th>Framework</th>
<th>Logo</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Django</td>
<td><img src="image" alt="Django" /></td>
<td><a href="https://www.djangoproject.com">Django</a></td>
</tr>
<tr>
<td>Ruby on Rails</td>
<td><img src="image" alt="Ruby on Rails" /></td>
<td><a href="https://rubyonrails.org">Ruby on Rails</a></td>
</tr>
<tr>
<td>Flask</td>
<td><img src="image" alt="Flask" /></td>
<td><a href="https://flask.palletsprojects.com">Flask</a></td>
</tr>
<tr>
<td>Pylons project</td>
<td><img src="image" alt="Pylons project" /></td>
<td><a href="https://www.pylonsproject.org">Pylons project</a></td>
</tr>
<tr>
<td>Sinatra</td>
<td><img src="image" alt="Sinatra" /></td>
<td><a href="https://www.sinatra.info">Sinatra</a></td>
</tr>
<tr>
<td>Laravel</td>
<td><img src="image" alt="Laravel" /></td>
<td><a href="https://laravel.com">Laravel</a></td>
</tr>
<tr>
<td>CodeIgniter</td>
<td><img src="image" alt="CodeIgniter" /></td>
<td><a href="https://codeigniter.com">CodeIgniter</a></td>
</tr>
<tr>
<td>web2py</td>
<td><img src="image" alt="web2py" /></td>
<td><a href="https://www.web2py.com">web2py</a></td>
</tr>
<tr>
<td>Symfony</td>
<td><img src="image" alt="Symfony" /></td>
<td><a href="https://symfony.com">Symfony</a></td>
</tr>
<tr>
<td>Zend Framework</td>
<td><img src="image" alt="Zend Framework" /></td>
<td><a href="https://framework.zend.com">Zend Framework</a></td>
</tr>
<tr>
<td>Mojolicious</td>
<td><img src="image" alt="Mojolicious" /></td>
<td><a href="https://mojojoel.com">Mojolicious</a></td>
</tr>
<tr>
<td>CherryPy</td>
<td><img src="image" alt="CherryPy" /></td>
<td><a href="https://www.cherrypy.org">CherryPy</a></td>
</tr>
<tr>
<td>Vaadin</td>
<td><img src="image" alt="Vaadin" /></td>
<td><a href="https://vaadin.com">Vaadin</a></td>
</tr>
<tr>
<td>PRADO</td>
<td><img src="image" alt="PRADO" /></td>
<td><a href="https://www.prado.com">PRADO</a></td>
</tr>
<tr>
<td>WT</td>
<td><img src="image" alt="WT" /></td>
<td><a href="https://www.intersol.com">WT</a></td>
</tr>
<tr>
<td>Phalcon</td>
<td><img src="image" alt="Phalcon" /></td>
<td><a href="https://www.phalconphp.com">Phalcon</a></td>
</tr>
<tr>
<td>Padrino</td>
<td><img src="image" alt="Padrino" /></td>
<td><a href="https://www.padrino-framework.com">Padrino</a></td>
</tr>
<tr>
<td>Stripes</td>
<td><img src="image" alt="Stripes" /></td>
<td><a href="https://www.stripesframework.org">Stripes</a></td>
</tr>
<tr>
<td>FuelPHP</td>
<td><img src="image" alt="FuelPHP" /></td>
<td><a href="https://fuelphp.com">FuelPHP</a></td>
</tr>
<tr>
<td>JBoss Seam</td>
<td><img src="image" alt="JBoss Seam" /></td>
<td><a href="https://www.jboss.org/seam">JBoss Seam</a></td>
</tr>
<tr>
<td>Merb</td>
<td><img src="image" alt="Merb" /></td>
<td><a href="https://merb.io">Merb</a></td>
</tr>
<tr>
<td>Silex</td>
<td><img src="image" alt="Silex" /></td>
<td><a href="https://silex.Router/">Silex</a></td>
</tr>
<tr>
<td>Yesod</td>
<td><img src="image" alt="Yesod" /></td>
<td><a href="https://yesodweb.com">Yesod</a></td>
</tr>
<tr>
<td>Vaadin</td>
<td><img src="image" alt="Vaadin" /></td>
<td><a href="https://vaadin.com">Vaadin</a></td>
</tr>
<tr>
<td>PRADO</td>
<td><img src="image" alt="PRADO" /></td>
<td><a href="https://www.prado.com">PRADO</a></td>
</tr>
<tr>
<td>WT</td>
<td><img src="image" alt="WT" /></td>
<td><a href="https://www.intersol.com">WT</a></td>
</tr>
<tr>
<td>Phalcon</td>
<td><img src="image" alt="Phalcon" /></td>
<td><a href="https://www.phalconphp.com">Phalcon</a></td>
</tr>
<tr>
<td>Padrino</td>
<td><img src="image" alt="Padrino" /></td>
<td><a href="https://www.padrino-framework.com">Padrino</a></td>
</tr>
<tr>
<td>Stripes</td>
<td><img src="image" alt="Stripes" /></td>
<td><a href="https://www.stripesframework.org">Stripes</a></td>
</tr>
<tr>
<td>FuelPHP</td>
<td><img src="image" alt="FuelPHP" /></td>
<td><a href="https://fuelphp.com">FuelPHP</a></td>
</tr>
<tr>
<td>JBoss Seam</td>
<td><img src="image" alt="JBoss Seam" /></td>
<td><a href="https://www.jboss.org/seam">JBoss Seam</a></td>
</tr>
<tr>
<td>Merb</td>
<td><img src="image" alt="Merb" /></td>
<td><a href="https://merb.io">Merb</a></td>
</tr>
<tr>
<td>Silex</td>
<td><img src="image" alt="Silex" /></td>
<td><a href="https://silex.Router/">Silex</a></td>
</tr>
<tr>
<td>Yesod</td>
<td><img src="image" alt="Yesod" /></td>
<td><a href="https://yesodweb.com">Yesod</a></td>
</tr>
<tr>
<td>Vaadin</td>
<td><img src="image" alt="Vaadin" /></td>
<td><a href="https://vaadin.com">Vaadin</a></td>
</tr>
<tr>
<td>PRADO</td>
<td><img src="image" alt="PRADO" /></td>
<td><a href="https://www.prado.com">PRADO</a></td>
</tr>
<tr>
<td>WT</td>
<td><img src="image" alt="WT" /></td>
<td><a href="https://www.intersol.com">WT</a></td>
</tr>
<tr>
<td>Phalcon</td>
<td><img src="image" alt="Phalcon" /></td>
<td><a href="https://www.phalconphp.com">Phalcon</a></td>
</tr>
<tr>
<td>Padrino</td>
<td><img src="image" alt="Padrino" /></td>
<td><a href="https://www.padrino-framework.com">Padrino</a></td>
</tr>
<tr>
<td>Stripes</td>
<td><img src="image" alt="Stripes" /></td>
<td><a href="https://www.stripesframework.org">Stripes</a></td>
</tr>
<tr>
<td>FuelPHP</td>
<td><img src="image" alt="FuelPHP" /></td>
<td><a href="https://fuelphp.com">FuelPHP</a></td>
</tr>
<tr>
<td>JBoss Seam</td>
<td><img src="image" alt="JBoss Seam" /></td>
<td><a href="https://www.jboss.org/seam">JBoss Seam</a></td>
</tr>
<tr>
<td>Merb</td>
<td><img src="image" alt="Merb" /></td>
<td><a href="https://merb.io">Merb</a></td>
</tr>
<tr>
<td>Silex</td>
<td><img src="image" alt="Silex" /></td>
<td><a href="https://silex.Router/">Silex</a></td>
</tr>
<tr>
<td>Yesod</td>
<td><img src="image" alt="Yesod" /></td>
<td><a href="https://yesodweb.com">Yesod</a></td>
</tr>
<tr>
<td>Vaadin</td>
<td><img src="image" alt="Vaadin" /></td>
<td><a href="https://vaadin.com">Vaadin</a></td>
</tr>
<tr>
<td>PRADO</td>
<td><img src="image" alt="PRADO" /></td>
<td><a href="https://www.prado.com">PRADO</a></td>
</tr>
<tr>
<td>WT</td>
<td><img src="image" alt="WT" /></td>
<td><a href="https://www.intersol.com">WT</a></td>
</tr>
<tr>
<td>Phalcon</td>
<td><img src="image" alt="Phalcon" /></td>
<td><a href="https://www.phalconphp.com">Phalcon</a></td>
</tr>
<tr>
<td>Padrino</td>
<td><img src="image" alt="Padrino" /></td>
<td><a href="https://www.padrino-framework.com">Padrino</a></td>
</tr>
<tr>
<td>Stripes</td>
<td><img src="image" alt="Stripes" /></td>
<td><a href="https://www.stripesframework.org">Stripes</a></td>
</tr>
<tr>
<td>FuelPHP</td>
<td><img src="image" alt="FuelPHP" /></td>
<td><a href="https://fuelphp.com">FuelPHP</a></td>
</tr>
<tr>
<td>JBoss Seam</td>
<td><img src="image" alt="JBoss Seam" /></td>
<td><a href="https://www.jboss.org/seam">JBoss Seam</a></td>
</tr>
<tr>
<td>Merb</td>
<td><img src="image" alt="Merb" /></td>
<td><a href="https://merb.io">Merb</a></td>
</tr>
<tr>
<td>Silex</td>
<td><img src="image" alt="Silex" /></td>
<td><a href="https://silex.Router/">Silex</a></td>
</tr>
<tr>
<td>Yesod</td>
<td><img src="image" alt="Yesod" /></td>
<td><a href="https://yesodweb.com">Yesod</a></td>
</tr>
</tbody>
</table>
Plugin

- Explicit extension points
- Static/Dynamic composition
- Low security (3rd party code)
- Extensibility and customizability
Client Side

HTML 5

JavaScript

CSS 3
JavaScript

• a programming language executed in the browser
  
  nothing to do with java

• JS files/libraries referenced by the web page
  
  like any other resource

• Can be inlined

• Dynamically manipulates the page Document Object Model (DOM) to alter page’s content, structure, and behavior
AJAX
Asynchronous JavaScript and XML
JS not so (well designed) easy
JS Frameworks
Case Study
Main Scenarios

• A **user** requests an article during normal operation and gets the rendered article HTML page.

• An **editor** saves an edited article during normal operation and the article is saved.
Introduction

A "LAMP" stack is a group of open source software that is typically installed together to enable a server to host dynamic websites and web apps. This term is actually an acronym which represents the Linux operating system, with the Apache web server. The site data is stored in a MySQL database, and dynamic content is processed by PHP.
Introduction

A "LAMP" stack is a group of open source software that is typically installed together to enable a server to host dynamic websites and web apps. This term is actually an acronym which represents the Linux operating system, with the Apache web server. The site data is stored in a MySQL database, and dynamic content is processed by PHP.
Qualities

• “Basic” implementation
  - Limited scalability
  - Single point of failure
  - Limited Security
Performance Tactics

• Control Resource Demand
  - *Increase the resource efficiency (caching)*
  - *Reduce overhead (pre-generate HTML from PHP)*

• Manage Resources
  - *Schedule resources (load balancer)*

https://www.digitalocean.com/community/tutorials/5-common-server-setups-for-your-web-application
Load Balancing

deploy many replicated instances of the server on multiple machines
Squid: Optimising Web Delivery

Squid is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. Squid has extensive access controls and makes a great server accelerator. It runs on most available operating systems, including Windows and is licensed under the GNU GPL.

Making the most of your Internet Connection

Squid is used by hundreds of Internet Providers world-wide to provide their users with the best possible web access. Squid optimises the data flow between client and server to improve performance and caches frequently-used content to save bandwidth. Squid can also route content requests to servers in a wide variety of ways to build cache server hierarchies which optimise network throughput.

Website Content Acceleration and Distribution

Thousands of web-sites around the Internet use Squid to drastically increase their content delivery. Squid can reduce your server load and improve delivery speeds to clients. Squid can also be used to deliver content from around the world - copying only the content being used, rather than inefficiently copying everything. Finally, Squid's advanced content routing configuration allows you to build content clusters to route and load balance requests via a variety of web servers.

"[The Squid systems] are currently running at a hit-rate of approximately 75%, effectively quadrupling the capacity of the Apache servers behind them. This is particularly noticeable when a large surge of traffic arrives directed to a particular page via a web link from another site, as the caching efficiency for that page will be nearly 100%." - Wikimedia Deployment Information.

Want to learn more?
Squid is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. Squid has extensive access controls and makes a great server accelerator. It runs on most available operating systems, including Windows and is licensed under the GNU GPL.
Caching + Load Balancing

FrontEnd (Apache)
Caching + Load Balancing

Squid (Caching) → Apache
NGINX Plus is the all-in-one application delivery platform for the modern web.

NGINX is the world's most popular open source web server and load balancer for high-traffic sites, powering over 300 million properties.

NGINX Plus adds enterprise-ready features for HTTP, TCP, and UDP load balancing, such as session persistence, health checks, advanced monitoring, and management to give you the freedom to innovate without being constrained by infrastructure.

LOAD BALANCER
Optimize the availability and uptime of apps, APIs, and services.

CONTENT CACHE
Accelerate your users' experience with local origin servers and edge servers.

WEB SERVER
Deliver assets with unparalleled speed and efficiency.

SECURITY CONTROLS
Protect apps using configurable security controls and authentication.

MONITORING & MANAGEMENT
Ensure health, availability, and performance with DevOps-friendly tools.

TRY NGINX PLUS FOR FREE
Caching + Load Balancing
Caching + Load Balancing

LoadBalancer (Squid) - Squid - LoadBalancer - Apache
Caching + Load Balancing

LoadBalancer (Squid) -> Squid -> LoadBalancer -> Apache

LoadBalancer

Squid

Squid

Squid

LoadBalancer

Apache

Apache

Apache
Performance Tactics

• Control Resource Demand
  - Prioritize events (deferred article updates)

• Manage Resources
  - Introduce concurrency (distributed database)
  - Schedule resources (load balancer)
  - Multiple copies of data and computations

https://www.digitalocean.com/community/tutorials/5-common-server-setups-for-your-web-application
Master/Slave

split a large job into smaller independent partitions which can be processed in parallel

Diagram: 
- Client
- Master
- Partition
- Slave A
- Slave B

Flow: 
1. Client
2. Assign to Slaves
3. Slave A
4. Slave B
5. Merge Results
Distribution + Replication

More reads than writes
Near-live updates (no strict consistency requirements)
Distribution + Replication

More reads than writes
Near-live updates (no strict consistency requirements)
Distribution + Replication

Data Sharding

Clear data separation (Article Name: A-B, C-D, etc..)
Distribution + Replication

![Diagram showing a load balancer directing traffic to a DB Master and a DB Slave, with 'Writes' and 'Reads' labels indicating traffic direction.]
Distribution + Replication

Load Balancer (Master)

Partition Logic (Sharding, Relication)

DB Master (Shard)

DB Slave

 Writes

 Reads
Visualization
Diagram of a system architecture:

- UI Page
  - Cache
  - Loader
    - Submit Logic
    - Static Resources
    - ArticleEdit
  - ArticleView
    - Cache
    - Parser
  - Skinning
  - Localization
    - Cache

- Backend
- Write Logic
- Read Logic
Security/Availability Tactics

• Prevent Attacks
  • Challenge Tokens (CSRF)
  • Validation (User) and Sanitization (SQL Injection, XSS)

• Resist Attacks
  • Maintain multiple copies of computations
  • Maintain multiple copies of data

• Recover from Attacks
  • DB Versioning (Recovery from data loss)
Additional Qualities
Extensibility

UI Page

Submit Logic

Loader

Static Resources

Sanitizer

ArticleEdit

Reads

Parser

Cache

Skinning

Localization

Cache

Cache

Cache

Backend

Static Resources
Extensibility

- UI Page
- Cache
- ArticleView
- Skinning
- Localization
- Static Resources
- Loader
- Sanitizer
- Submit Logic
- ArticleEdit
- Backend
- Parser
- Hook Engine
- External Module
- Notify
- Register Callback
- Reads
- Writes
Configurability/Customizability
Configurability/Customizability

Global Variables and Configurations

Submit Logic

Article Edit

Sanitizer

Article View

Parser

Skinning

Localization

Cache

Loader

Static Resources

Cache

Cache

Cache

Cache
Process View
A user requests an article during normal operation and gets the rendered article HTML page.
Write/Edit Article

An **editor** saves an edited article during normal operation and the article is saved.
Write/Edit Article

Error Page

Write Conflict

Update Job

Create Edit Form

Render Content

Submit Logic

User Authorized

Article Edit

LoadBalancer Select DB
Summary

- Work incrementally
- Use different architectural views
- Start the design from the domain model and go up in the layers
- Use frameworks whenever possible
- Each design decision has a rationale (hoisting)